

# **Machine Learning for Prediction and Analysis of Public Financial Data**

**Submitted To**

**Dr. Haris Vikalo  
State Street Corporation**

**Prepared By**

**Rajat Ahuja  
Casey Hu  
Seungjun Lee  
Aaron North**

**EE464 Senior Design Project  
Electrical and Computer Engineering Department  
University of Texas at Austin**

**Fall 2019**

## CONTENTS

<b>TABLES.....</b>	<b>iii</b>
<b>FIGURES.....</b>	<b>iv</b>
<b>EXECUTIVE SUMMARY .....</b>	<b>v</b>
<b>1.0 INTRODUCTION.....</b>	<b>1</b>
<b>2.0 DESIGN PROBLEM STATEMENT .....</b>	<b>2</b>
<b>3.0 DESIGN PROBLEM SOLUTION.....</b>	<b>3</b>
<b>3.1 EDGAR Data Scraper .....</b>	<b>4</b>
<b>3.2 Stock Data Scraper .....</b>	<b>6</b>
<b>3.3 K-Means Clustering.....</b>	<b>7</b>
<b>3.4 Recurrent Neural Network .....</b>	<b>9</b>
<b>3.5 Web Application.....</b>	<b>11</b>
<b>4.0 DESIGN IMPLEMENTATION.....</b>	<b>12</b>
<b>4.1 Formulation of Design Specifications.....</b>	<b>12</b>
<b>4.2 Obstacles for Implementation.....</b>	<b>13</b>
<b>5.0 TESTING AND EVALUATION .....</b>	<b>15</b>
<b>5.1 Data Scraper.....</b>	<b>15</b>
<b>5.2 Recurrent Neural Network .....</b>	<b>17</b>
<b>6.0 TIME AND COST CONSIDERATIONS.....</b>	<b>17</b>
<b>7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN.....</b>	<b>18</b>
<b>8.0 RECOMMENDATIONS.....</b>	<b>19</b>
<b>8.1 Obtaining Data .....</b>	<b>19</b>
<b>8.2 Understanding Correlations .....</b>	<b>21</b>
<b>9.0 CONCLUSIONS .....</b>	<b>22</b>
<b>REFERENCES.....</b>	<b>24</b>
<b>APPENDIX A – SPECIFICATIONS TABLE .....</b>	<b>A-1</b>
<b>APPENDIX B – USER INTERFACE SCREENSHOTS .....</b>	<b>B-1</b>
<b>APPENDIX C – RANDOMLY GENERATED DATA .....</b>	<b>C-1</b>

## TABLES

1	Example Section From Combined DataFrame .....	5
2	Walmart Quarterly Report Data, Q4 2018 .....	16
3	Contents of Each Cell in the Row for Net Sales .....	17
4	Specifications Table .....	A-2

## FIGURES

1	Block Diagram of Final Solution .....	3
2	Representation of Company X Dataset for RNN Processing .....	4
3	Elbow Curve of Inertia in K-Means Clustering .....	8
4	Prediction Page of Web Application.....	B-2
5	Clustering Page of Web Application .....	B-3
6	Reframed Randomly Generated Data .....	C-2

## EXECUTIVE SUMMARY

The problem of predicting future trends in the stock market has long been a goal for both individual investors and financial services companies. The aim of our project was to utilize the public data stored on the SEC's financial database, EDGAR (Electronic Data Gathering, Analysis, and Retrieval), to create a web application that would aid users in making wise investment decisions. In the past, artificial neural networks have been used to take on the task of stock market prediction by training on past stock data and outputting whether one should buy or sell a given stock. However, the vast volume of company reports featuring both qualitative and quantitative data present in the SEC's database has received little attention in this regard. The work we have done this past year has centered around attempting to create a machine learning tool using EDGAR data to predict the future value of public companies queried by the user.

The first step in our preparation phase was to identify the types of reports we would be focusing on among the many types of financial documents present in EDGAR. We decided to narrow the scope of our project to only examining the 10-Q forms submitted to the SEC between 2008 to the present. Our reasons for limiting our dataset to just this set of documents are explained in detail in Section 3.1 of this document, which goes over our data gathering process. Quarterly reports are an obvious candidate for a tool used to aid investment decisions because they contain comprehensive information, including large amounts of numerical data, that informs shareholders on the overall performance of the company in that quarter. With our data source determined, our next step was to obtain our targeted data.

Without access to an API that could simplify the process, we had to create a scraper from scratch in Python. Due to difficulties encountered with this step in our project, we chose to limit the starting data in our model to only data from the following significant tech companies: Apple, AMD, Amazon, Cisco, Google, IBM, Intel, Nvidia, and Texas Instruments. Our project thus operated on the expectation that the model would only perform well if a much larger amount of data was fed into it. Regardless, we decided to move on to obtaining the historical stock price data we would need to label each data point in our neural network.

Finding the past stock data for a publicly traded company is a far simpler matter. For this task, we used the Alpha Vantage API, which contains day-to-day stock information for any queried company from 20 years ago to the present, which more than met our needs. We utilized the stock data to perform K-means clustering on the day-to-day data from Alpha Vantage using all data from our nine companies from 2008 to the present. The clusters produced by this algorithm can serve as valuable information to investors by themselves, but we further used them to aid our machine learning model. When making predictions on a particular company, we would incorporate stock data from other companies which have similar growth trends to the targeted company (i.e. companies within the same cluster) to our dataset. This ended up significantly reducing overfitting of our prediction model. Thus, the day-to-day stock data became a useful asset in improving our predictor, a recurrent neural network.

We selected a recurrent neural network (RNN) with long short-term memory (LSTM) cells to accomplish prediction, as this form of neural network is well suited for time series analysis and is commonly used by existing models in the field of stock market prediction. We trained on all

data from 2008 to the second quarter of 2019; in our dataset, each row contained the data for a company in that quarter's 10-Q form in addition to the stock price of all companies within that cluster shortly after the report was filed. We then validated on the final row of the data. Our predictions for this last row contained the correct trend five out of nine times, that is, the direction (positive or negative) of the predicted change in stock price matched the actual direction of the change in stock price for five out of our nine companies. Considering the limited amount of data we had to train our model, this level of accuracy is reasonable. With the model constructed, our final step was to incorporate our data and machine learning models into an application that could be expanded with additional data to increase the accuracy of future predictions.

Our final product was a web application with three main sections: a clustering page that could perform k-means clustering on a group of companies using their historical stock data and output the results to the user; a page for users to upload financial data of the same format as our existing data; and a page that would utilize both the clustering algorithm and the RNN on our back end to make predictions on a queried company based on the data supplied by the user. This format is highly scalable, as the complexity of the RNN could be increased just by supplying more rows of data (the most obvious additional source of data would be annual reports) and by adding more features that could be correlated with stock price.

Ultimately, the capabilities of our application are more limited than we would have hoped. The large granularity of the data from EDGAR (a maximum of four points of data per year) as well as the lack of any universal standard for how 10-Q files were organized over the years led to a shortage of data with which to train our model. Fortunately, the ability of users to supply their own data and decide on the number of features they want to include per company means our product has great potential to substantiate its predictive power in the future. Additionally, the creation of a more refined scraping tool that would allow users to easily search through the entirety of EDGAR would greatly benefit this and similar projects. We believe that our product provides a valuable starting point for the creation of a tool that would be a highly informative provider of financial advice to individual investors and companies alike.

## **1.0 INTRODUCTION**

This document will describe the engineering problem at the core of our project and the different modules of the solution we created through our work in the past year. The goal of this project was to utilize quarterly reports stored in EDGAR, the SEC's public database of financial information, to predict future trends in the stock market. Our final product combines user-inputted data with our own starter data to train a recurrent neural network, which in turn, outputs a prediction that is displayed on our web application. This product is aimed at users who can supply their own financial data, and will aid them in making wise investment decisions regarding their company of choice.

The contents of this document, which lays out every aspect of our project and the motivation behind our design decisions, are broken up into the following sections. In Section 2, we will state the goals and motivation behind our project and the requirements our web application aimed to fulfill. In Section 3, we will clearly describe the subsystems of our solution and the technical details behind each one, as well as how they work as a whole to meet the specifications in the problem statement. Each subsection will cover one aspect of the design: the data scrapers, the clustering algorithm, the RNN model, and the front end of the web application. In Section 4, we will discuss how we adjusted our vision of the design solution due to difficulties with scraping data and tuning the neural network, as well as how our eventual implementation differed from alternative ideas we had early on. In Section 5, we will review how we tested each module to achieve a satisfactory level of accuracy, and whether or not we were able to satisfy our initial expectations. In Section 6, we will cover how time constraints affected our solution, especially with regards to the time given to data scraping. In Section 7, we will address ethical concerns regarding the usage of our stock price predictor. In Section 8, we will describe ways our design could be improved, particularly how we would create a better machine learning model, with more suitable data and greater attention given to outside factors that could affect a company's stock price. Finally, we will conclude by summarizing our findings and achievements throughout the past year and evaluating the extent to which our solution met the project requirements.

## 2.0 DESIGN PROBLEM STATEMENT

The impetus for creating our financial analysis model was the relatively unexplored wealth of data stored in EDGAR. EDGAR (or the Electronic Data Gathering, Analysis, and Retrieval system) is a financial database maintained by the SEC. All companies that conduct business within the United States are required to file certain documents, including periodic reports and registration statements, through this database [1]. Currently, there are few applications that make use of any of the publicly available data stored in EDGAR; leaving this well of useful, widely varying data untapped. As data becomes increasingly vital to the expansion of new technologies, EDGAR will inevitably be exploited for its potential to unlock unexplored frontiers of the financial prediction world. Our goal was to supply one of the first solutions to prove the existence of that potential. In order to reach this goal, we were faced with the challenge of utilizing data within EDGAR to make predictions for stock prices in the future.

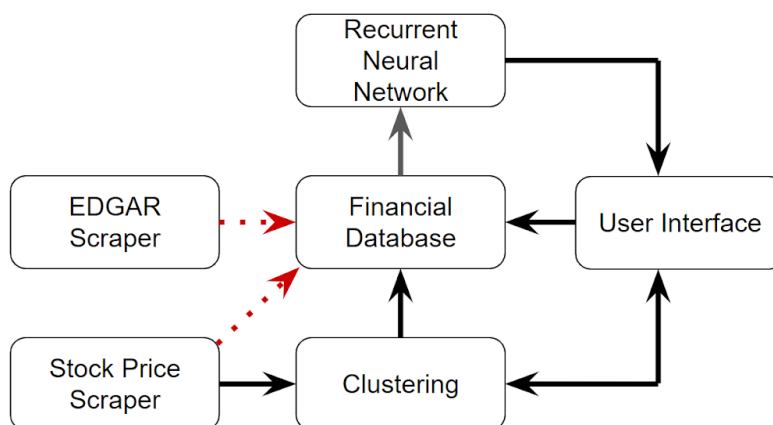
Given this challenge, we established a set of high level requirements for our solution, and chose to create a web application that could meet those expectations. First, our solution must be able to examine sets of financial data to predict stock prices one quarter in the future. This examination took the form of both a recurrent neural network (RNN) and a K-Means clustering algorithm. Users of our solution must be able to upload financial data to our application. To ensure these data conform to our predetermined format, we must also guide users through the upload process, required features, and specifications (ex. numerical values in millions of dollars, except per share data). Finally, our application must be simple to use. This requires a sleek look and simple navigation. These high level requirements are further deconstructed in the specifications table listed as Table 4 in Appendix A. As shown in this table, we included a starter set of data containing financial information for nine businesses spanning the last decade that can be run through our RNN. With this starter set, all users are able to make an initial set of predictions without uploading their own data. Our back end subsystems are all written in Python and employ several useful libraries such as Tensorflow and tslearn. We have chosen to use Javascript for both our front and back ends of the web application itself to simplify the process of pipelining data. Finally, the user interface was designed prior to deployment, in order to be simple to navigate and understand, with a focus on minimalist page design and informative structure. By



outlining these requirements, we were able to appropriately guide the development process and ensure that our application meet the goal we set for ourselves.

### 3.0 DESIGN PROBLEM SOLUTION

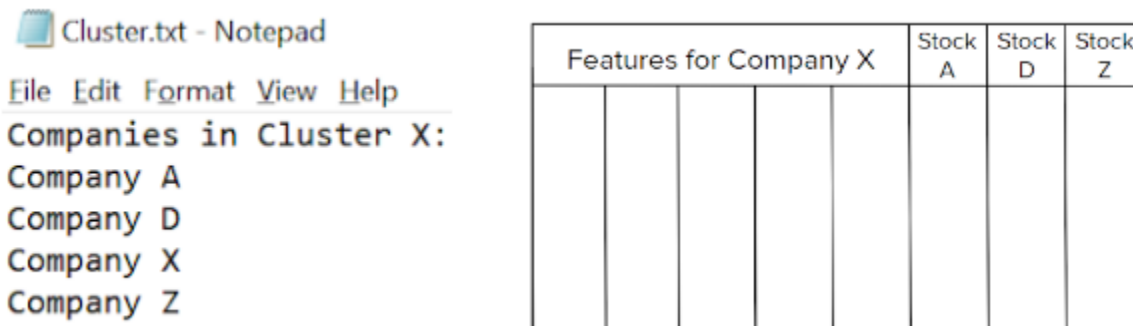
Our final solution is a web application that is able to store and analyze financial datasets to produce stock price predictions one quarter into the future. As shown in the block diagram in Figure 1 below, the application consists of several subsystems that work in tandem to make these predictions. First, we created two data scrapers that pull data from EDGAR and stock prices from the Alpha Vantage API, respectively.



**Figure 1. Block Diagram of Final Solution**

These scrapers provided the initial starter set of nine companies that compose our financial database as shown by the red arrows in Figure 1. The nine companies include Amazon, AMD, Apple, Cisco, Google, IBM, Intel, Nvidia, and Texas Instruments, and they were chosen for their relevance in the tech industry. The stock price scraper also dynamically feeds data into the K-Means clustering module based on user input as shown by the black arrows in Figure 1 above. The clustering module utilizes a modified K-Means algorithm to group companies together based on how similar or dissimilar their stock prices change over time. The results of clustering are used to supplement the predictions made by the RNN. When a user queries the web application for a stock price prediction of a specific company, the clustering module is triggered and creates a list of other companies (stored as a .txt file on the back end) that are clustered together with

that initial company. The financial database is then modified to include the stock prices of that list of companies as additional features to be used by the RNN, as shown in Figure 2 below.



Features for Company X					Stock A	Stock D	Stock Z

**Figure 2. Representation of Company X Dataset for RNN Processing**

Users also have the ability to upload data to our database for companies that are not included in the starter set. Once the financial database is in the state desired by the user, they can ask the application to make a prediction for a specific company. This triggers the RNN which feeds the database through a machine learning model to produce the desired prediction. This prediction is displayed to the user through our user interface.

The remainder of this section goes into further detail describing each of the modules shown in Figure 1 on the previous page. For each module, we will discuss our specific method for accomplishing the objective for that module. Additionally, each module faced unique challenges that both hindered development and provided inspiration for the final solution. Lastly, we will review whether or not we believe each module fulfilled its purpose for meeting our specifications listed in section 2.0.

### 3.1 EDGAR Data Scraper

The EDGAR quarterly report data that serves as the backbone of our product was obtained by a scraper written from scratch in Python. This scraper was used in conjunction with the stock data scraper to form the dataset used by the RNN. We used Pandas [2] DataFrames to store the data in matrices, an example of which can be found in Table 1 below. The scraper read in data from all quarterly reports of each of our nine starter companies from the first quarter of 2008 to the

second quarter of 2019. We would have preferred to have far more rows of data per company for our training set, but it was not possible to achieve finer granularity in our data from the sources available.

**Table 1. Example Section From Combined DataFrame**

Company	Year	Quarter	Revenue	Cost of Revenue	Net income	Earnings per share	...
Apple	2018	1	88293	54381	20065	3.89	...
Apple	2018	2	61137	37715	13822	2.73	...
Apple	2018	3	53265	32844	11519	2.34	...

Obtaining well-formed data from all documents scraped, over 300 in total, was a major undertaking. Scraping was done by writing functions in Python to access the URL of the document corresponding to an inputted company name, year, and quarter, and searching through each HTML table in the document until the targeted search terms were all found. HTTP requests to the SEC database were sent using the Requests library [3], while the documents were parsed using the BeautifulSoup library [4]. These search terms, which represent the features of our machine learning dataset, were: revenue, cost of revenue, net income, earnings per share, current and total assets, current liabilities, shareholders' equity, cash lost through depreciation, cash generated through operating activities, and cash at the end of the period. These terms were chosen because they were all possible metrics of a company's activity during the quarter, and were all consistently available within each report in each quarter. The companies we chose were: Amazon, Apple, AMD, Cisco, Google, IBM, Intel, Nvidia, and Texas Instruments. They were chosen for their size, name recognition, and relevance to the fields of computers and computer components.

There are many difficulties associated with obtaining data found in EDGAR files that made scraping data from more companies too time-consuming. The way in which they are formatted greatly benefits those that choose to manually review a single file; scraping large quantities of data is far more difficult. Since the early 2000s, 10-Q documents have been stored as HTML documents, making scraping with libraries such as BeautifulSoup possible, but the formatting

has changed drastically, with all companies often having notable differences in their document structure from 2008 to the present. This made the process of attempting to parse the same key phrases from different companies at different times unexpectedly difficult. Nonetheless, we achieved our goal from the design problem of making use of EDGAR data; the next step was to label each of the feature vectors with the target label: the stock price of the company around the time of the report’s filing.

### **3.2 Stock Data Scraper**

Obtaining historical stock data for a given company was a far simpler process compared to scraping EDGAR data. Using a free license for the Alpha Vantage API [5], we were able to obtain any publicly traded company’s stock data from any business day in the past twenty years, which was more than enough for our purposes. We labeled each vector of data from a quarterly report with the adjusted closing stock price around the time that the report was filed, which takes into account any corporate actions that might affect the value of a stock [6]. This allows us to discover basic correlations between the various features in the 10-Q forms and a company’s stock price, but it is important to remain aware that the low granularity of the EDGAR data (only once per quarter) remains a limiting factor in creating a dataset to approximate continuous data, which is ideal for an RNN.

Another key point to recognize is that only a small fraction of the Alpha Vantage stock data is utilized directly in the data. The values that we obtain from the API are the adjusted closing stock price, which is the “true” value of the stock compared to the closing price, which doesn’t take into account actions such as stock splits that have major effects on the actual worth of each share. These values are calculated from the last transaction before the market closes, and there are approximately 250 values per year compared to the four per year used in labeling the EDGAR data. Thus, the vast majority would be unused if we simply discarded the remaining values. However, we chose to apply all of the stock data for each company in a related supplementary task: clustering the companies by their historical stock data. This usage, which we will explain fully in the following section, allowed us to use correlations between companies’ stock histories without adding too much information to the DataFrame used in training the RNN.

This allowed our model to accomplish both correlation and prediction while successfully reducing the possibility of overfitting.

### **3.3 K-Means Clustering**

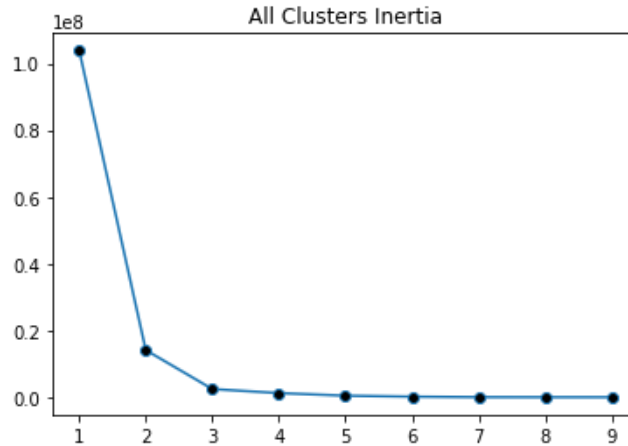
Our initial focus for approaching this design problem centered around the idea of correlation. While we eventually narrowed our goal to prediction, we felt that correlation was still a key component of our design solution. Consequently, we explored methods of discovering correlations within a vast dataset. Correlation can be implemented in a variety of ways that range from simple linear regressions to more complicated machine learning models. We chose to work with unsupervised clustering. Clustering partitions a dataset into smaller groups, or clusters, that are theoretically correlated in some way. Many clustering algorithms work on both small and large datasets. At this stage of design, we expected our solution to begin with a smaller dataset and evolve to a much larger dataset as the user base grew. Additionally, we were unsure of the number of features we may integrate into our correlation model. We decided to initially base our clustering model on the stock price feature considering that our project solution focuses on stock price prediction. Yet, we remained open to the possibility of correlating using other features as well. By focusing on clustering, we developed a model that could utilize multiple features, operate properly on our initial smaller dataset, and perform even better with larger datasets in the future.

After considering numerous clustering algorithms such as DBSCAN and Mean-Shift, we selected K-Means. K-Means is a popular clustering algorithm that does not require heavy processing power to implement. The algorithm clusters a dataset by creating ‘cluster centers’ and measuring the distance from each data point to its center. It then adjusts these centers iteratively to optimize the clusters based on overall distance, or ‘inertia.’ However, our dataset consists of features over time, which is a temporal series, not single data points. This complicates the idea of what the ‘distance’ between two data in our dataset represents. In order to properly cluster our dataset, we defined the distance between two time series as the sum of the euclidean distances between them at each x-value. For example, the euclidean distance, with respect to stock price, between Company X and Company Y on October 31st, 2008 is the difference between their stock price on that day. Summing these differences over the entire time series using

$$\sum_{i=0}^n \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \quad (1)$$

to find the inertia allowed our clustering to have a consistent distance value with which to work, regardless of which features ended up in the final clustering dataset.

Another crucial characteristic of K-Means clustering is the choice of how many clusters are used to partition a dataset. Many clustering algorithms dynamically determine the number of clusters that a dataset should be grouped into. However, those algorithms typically require a lot of time and processing power to implement, often without a guarantee that the algorithm will converge successfully enough to form clusters. K-Means requires a predetermined number of clusters,  $K$ , to function. Discovering the optimal number of clusters requires running the K-Means algorithm to calculate the inertia for every possible number of clusters (or a small, representative subset for large  $n$ ). Because we had nine companies in our initial dataset, this resulted in the graph shown below in Figure 3.



**Figure 3. Elbow Curve of Inertia in K-Means Clustering**

This graph is known as an elbow curve, because after a certain point the difference in accuracy (with an inertia of zero meaning 100% accuracy) becomes increasingly insignificant. When K-Means is run with a single cluster, it means that every data point is in the same group. Thus, the inertia is at its highest possible value. When K-Means is run on  $N$  data points with  $N$  clusters, it

means each data point is in its own group. It follows that the cluster center of each cluster is the data point itself. In this case the inertia is zero. The optimal inertia is then found at the elbow point of our curve because the inertia has decreased significantly and is starting to converge towards zero. In Figure 3 on the previous page, the elbow lies between two and four clusters because after four clusters the change in inertia becomes relatively low. After running the K-Means algorithm again for two, three, and four clusters, we visually examined the results to verify which K, number of clusters, best represents our dataset. After this visual check, we determined that four clusters is the optimal number for our project.

### **3.4 Recurrent Neural Network**

In order to best correlate all the data scraped from EDGAR and to make a future stock prediction of companies, we built a recurrent neural network using the Keras API. Recurrent neural networks are useful for data analysis related to change over time. While neural networks, in general, correlate large datasets, recurrent neural networks add the dimension of time to that analysis. Our neural network is a sequential model with an input layer, a hidden layer, and a dense layer. Due to the nature of our application, the number of cells in the input layer is dynamic. Each time a user requests a stock price prediction for some company, our web application re-builds a model based on a modified version of that company's data. The modified dataset includes the features of that specific company, as well as the stock price of other companies based on the results of the K-Means clustering algorithm described in the previous section. Thus, the number of input variables can fluctuate based on several factors determined by the users of the application. The cells in the hidden layer are Long Short-Term Memory (LSTM) cells. LSTM cells filter data through the lens of time. When temporal data is run through LSTM cells, older data is either given less weight or forgotten entirely. In the context of stock market analysis, this is incredibly useful. Intuitively, stock prices from 10 years ago are much less likely to provide insight into tomorrow's stock price than the insight provided by today's stock price. These LSTM cells were set to have memory length of one, which means they placed highest significance on data from the most recent quarter. In this way, LSTM cells are able to incorporate that intuition into our model. We chose to place twelve LSTM cells in the hidden layer. This optimized the model's ability to provide temporal insight while avoiding severe overfitting. Lastly, the model consists of one cell in the dense layer. This layer is the output layer

that holds the future stock prediction. In alternate versions of the model we had more cells in the dense layer that provided predictions for multiple companies at a time. In conception, this idea seemed useful because it allowed us to create a single model that would not need to be rebuilt. However, in implementation this approach encountered severe overfitting and never converged to satisfying results.

The last but most important part of building a recurrent neural network is shaping the input vector. Due to the limited number of rows we had available with the data from EDGAR, we had to reduce the number of variables per row in the input vector that was being fed to the RNN to get a reasonable error on the predictions. At the same time, the input vector had to contain the data of the last quarter and the current quarter to enable the supervised learning in the LSTM cells. Thus, our code dynamically forms an input vector that contains rows with the current and a past quarter's dataset. The dataset consists of the 12 features of the company that the user selected and the stock prices of other companies in the same cluster. As a result, our input vector is dynamically formed to have 2 dimensions with varying number of columns in each row.

Although we did not build a model and save it in the back-end of our application, our dynamically built models yielded similar results each time we ran our application. For testing purposes, we trained on all the data we had except quarter 3 of 2019 and verified that we had on average 22% error with the 9 companies in our back-end. We also compared the trends of stocks from 2019 quarter 2 to quarter 3. These trends were checked each time we altered the parameters of the RNN. After several attempts, we had to settle with 5 out of 9 companies' trends matching. This resulted from the setting that yielded 22% average error as well. Ideally, more data and more memory in the LSTM cells would improve the model and decrease the error. Also, we learned that there are many unpredictable causations that affects the future stock prices. However, we have learned to make correlations among data and to compile and train a RNN from scratch using our own data in our project. After several testing and evaluations, we concluded that the model with 22% average error was the best one we can build with our resources and moved onto integrating this algorithm in the web application.



### 3.5 Web Application

In order to best facilitate end-user interaction with our machine learning scripts, we created a web application (displayed in Figures 4 and 5 in Appendix B) as our product interface. Our web application is a full-stack JavaScript application, meaning that both the client-side (front end) and the server-side (back end) code are written in JavaScript. We chose to use only JavaScript as we wanted to emphasize code reuse and sharing between the front end and back end. NodeJS was specifically chosen as our server-side framework for its vast open-source development toolset, npm, while ExpressJS was chosen as our server framework to run on NodeJS mainly because of its ubiquity, widespread community support, and ease-of-use in that role.

In lieu of connecting a database to our web application's back end, we store all of our data in the form of CSVs in our project's file directory. One of our reasons for doing so is that our data solely exists in the form of CSVs and there was no added benefit to storing these CSVs in a database when they could provide full functionality by existing in our file directory. Connecting a database to our back end would be both computationally expensive and add unneeded complexity to the back end.

Although our back end was written with JavaScript, our various machine learning algorithms (K-Means and RNN) were both written in Python. In addition, the scripts we wrote to combine user CSVs with our system's aggregate CSV and to scrape the Alpha Vantage API were also written in Python. In order to run these Python scripts from our NodeJS back end, we used NodeJS's built-in `child_process` module, which provided us the ability to spawn child processes asynchronously. This means that while our NodeJS main process was running the web application, our NodeJS child processes could concurrently run any of the Python scripts needed. These child processes also allowed us to pass JSON (JavaScript Object Notation) to any of the Python scripts needed and render JSON output from these scripts to the user.

Our web application has four main components that users can utilize. Most importantly, users can predict the future stock price (presently of 2019 Q4) for any particular company that has its financial data uploaded to our back end. To facilitate this process, users can upload CSVs with financial data for a particular company; users must first make sure the features and specifications

of their CSVs match our data template. Users can also scrape the historical stock prices of any company that they choose (via the Alpha Vantage API), provided that they have the company's stock symbol at hand. Finally, users can cluster the companies for which our back end has data, to see which companies have stock prices grow and decay (in a sense, trend) similarly.

## **4.0 DESIGN IMPLEMENTATION**

Arriving at our final design solution was not a straightforward, linear process. Going into this project, the team lacked both expertise and direction. None of our team members were particularly familiar with the relevant fields of data science and finance. In addition, the core goals of the problem as they were initially presented to us were vague, and the form the final deliverable would take was unclear, as State Street – prior to pulling out of the project – decided to leave everything mostly open-ended. The main struggles of our team in the first half of the year centered around understanding the problem statement, researching topics we lacked knowledge in, and coming up with a concrete course of action. In the second half of the year, after State Street's withdrawal, we faced challenges regarding obtaining data and creating a model despite a lack of data. Ultimately, we were able to implement a working web application that combines our back end starter data with user-inputted data to output a reasonably accurate predicted stock price. Sections 4.1 and 4.2 goes over our design and implementation process, and how we were able to overcome these difficulties to create a suitable prototype with great potential for further development.

### **4.1 Formulation of Design Specifications**

Before beginning to work on creating a design solution, we had to first fully define our own goals and expectations for this project. Our initial instructions were only to analyze EDGAR to discover correlations in financial data, but lacked concrete goals or specifications. Thus, we had to figure out two things: First, what were we ultimately going to “discover” from the data? Second, which files within EDGAR were we going to use? We settled on an answer for the first question after some brainstorming: We would attempt to use the data to predict future trends in the stock market. This matched with the initial stated purpose of this project, to create a tool to aid State Street in providing investment advice, as successful implementation of such a predictor would obviously be a valuable asset to any investor. The stock price of a company is also clearly

related to the performance of that company, which quarterly reports are meant to be a comprehensive analysis of, so the two metrics seemed to be sensible sets of data between which to discover correlations.

Coming up with an answer to the second question, relating to the obtaining of our financial data, was more difficult. There are over 160 types of forms stored in the EDGAR database, and examining or attempting to understand them all with no financial expertise would be infeasible for the time we had. After dividing up the task of researching the major forms between our team members, we realized that the vast majority of the forms had very specific usages, and the only forms that were universally submitted on a regular basis were the 10-K and 10-Q forms, or the annual and quarterly reports, respectively. These forms also contained numerical measurements of the company's performance during the listed period of time, which was perfect for our other goal of using machine learning to identify trends within and between companies. However, getting this data in a workable format ended up being the main challenge, and a major limitation to our application as a whole. Without an API to parse the data through function calls, we decided our only choice was to write a scraper from scratch to scrape 10-Q forms from EDGAR. Despite being the source of continuing future struggles, this scraper was the answer to our question of what data we would obtain from EDGAR.

## **4.2 Obstacles for Implementation**

The obstacles that we encountered while implementing our newly formulated design solution centered around the limitations of our data scraper, which is described in Section 3.1. The following difficulties with the formatting with reports limited our ability to obtain more data, causing us to only have quarterly reports from nine companies in our starter data:

- The ordering of terms within the document would change from one year to the next, and the exact wording could vary as well.
- The formatting between two companies could vary greatly, even if it was something as simple as saying "Net sales" or "Total revenue" instead of just "Revenue".
- The data that we wanted wasn't always located on the same row of the table as the label, for example "Earnings per share" would appear by itself in a table row, and then the value for diluted earnings per share would be found below, labeled only with "Diluted".

- Unexpected additions in a report, such as a second row for revenue to account for the temporary acquisition of another company, could confuse the scraper.
- Documents could have multiple “invisible” table cells that were undetectable until the scraper attempted to parse them and ended up hitting a soft limit for the number of tables it would check for search terms.

Because we realized the level of manual tuning required to make the scraper obtain the right information would be too large of a time commitment, we decided to leave the task of scraping the desired data to the user. That is, if a user wanted to predict the future stock value of a company not on our list, the burden was on them to provide the EDGAR data to input into the RNN. We also left the decision of whether or not to utilize other sources of data, whether they were annual reports found on EDGAR or data from Google Trends, to future groups that decide to refine our application (our recommendations are discussed in detail in Section 8). As a result of not being given proprietary data from State Street and the difficulty of scraping our own data from scratch, our expectations for the accuracy of the prediction produced by our model were low. Despite understanding that the size of our dataset was orders of magnitude too small, we decided to proceed with creating a recurrent neural network, even if it ended up being just a proof of concept.

The other main obstacle we faced during implementation was the difficulty associated with training our model with such little data. Tuning the RNN seemed like an impossible task early on; we could only validate our model on the last point of data, the third quarter of 2019, since there was so little data we wanted to keep as much of it as possible. Initially our error rates were incredibly high, somehow predicting stock prices in the tens of thousands of dollars despite previous values only being in the hundreds. It seemed at first that there was no hope of achieving predictions even within the same order of magnitude. We realized that we needed to simplify our model; with our lack of data, having too many features present would only cause overfitting. As described in Section 3.3, we ended up using the Alpha Vantage stock data to cluster companies and cut down on the size of our DataFrame, which greatly improved our validation error. As a result, we were able to partially overcome our earlier difficulties with data to create a predictor that produced reasonable results.

While integrating each of our models ended up being mostly straightforward, as outlined in Section 3.5, one major design decision we had to make was choosing how to incorporate the stock data into the model. With one row of data per quarterly report, we only had 35 rows total, while there were thousands of values available per company of the historical stock price. The surprising decision we ended up making was to *not* use the vast majority of the data in the DataFrame for our RNN. Two reasons guided this decision: First, such a huge quantity of data would end up overshadowing our EDGAR data, and one of the major goals of the project was to find a way to utilize EDGAR data in financial analysis. If we had just used past stock data to predict future stock values, we would be simply repeating what many others had done in the past, contributing no new findings. Second, we were uncertain how we would stretch out the 35 rows of EDGAR data to match up with the thousands of rows of stock data. We could have simply repeated the features from the quarterly reports for every stock value that matched up with that quarter, but we ultimately considered that a misleading use of data that would also likely lead to overfitting. The data from the quarterly reports does not contain constant values repeated every day for that quarter; they are a cumulative result of all company activity during the entire quarter. Thus, we decided not to add the entirety of the stock data to our training data. This was a good choice as it turned out, as we were able to show through our project the potential for EDGAR data to be applied to stock market prediction.

## **5.0 TESTING AND EVALUATION**

In order to test and evaluate our application, we physically went through the website and made sure that all the scripts were running when we prompted them to. Before putting the whole system together, we had to make sure that two big modules were functioning correctly individually. These two methods that enabled our application to gather data and correlate data were data scraper module and the RNN module. In this section, we will go over some of the major functions that were tested for these modules.

### **5.1 Data Scraper**

To test the accuracy of the scraper, we chose a company, a year, a quarter, and a list of search terms we wanted to find. We then compared the value returned by the function to the values in

that column of the table in the document. This is done using the wrapper function `get_html_tables`. A successful test run of the function is shown in Table 2 below, where a quick comparison shows that selected values from the table in the document (above) match the values inserted into a row in the DataFrame (below) using the scraper. The search terms supplied were ‘Net sales’, ‘Total revenues’, ‘Cost of sales’, and ‘Operating income’, and the leftmost values were grabbed by default. This form of testing can be repeated for any company and any year or quarter that company filed its quarterly reports in HTML format. Reports filed within the past decade could be guaranteed to be in HTML format, so this sort of simple comparison could be done for any company within our timeframe, and it would be easy to verify that the results obtained matched the values in the document.

**Table 2. Walmart Quarterly Report Data, Q4 2018**

		Three Months Ended October 31,				
		2018	2017			
<i>(Amounts in millions, except per share data)</i>						
<b>Revenues:</b>						
	Net sales	\$ 123,897	\$ 122,136			
	Membership and other income	997	1,043			
	Total revenues	124,894	123,179			
<b>Costs and expenses:</b>						
	Cost of sales	93,116	91,547			
	Operating, selling, general and administrative expenses	26,792	26,868			
	Operating income	4,986	4,764			
	Year	Quarter	Net sales	Total revenues	Cost of sales	Operating income
Walmart Inc. 2018-4	2018	4	123897.0	124894.0	93116.0	4986.0

The function was primarily tested and debugged by printing the value of variables as they were retrieved, and comparing them to the expected values. One difficulty we encountered was parsing the correct value within a row. Table 3 below shows the actual contents of each cell in the row labelled Net sales, where we find that unexpected padding, or whitespace, can be found in some of the cells. This whitespace is the Unicode character ‘\xa0’, otherwise known as a no-break space character. This unusual character was found in many documents and was undetectable with the usual print function in Python. However, by using the built-in ‘repr’ function, even invisible whitespace characters become visible. Our function used regex to only

grab cells containing numerical values to prevent non-numerical symbols and text from entering our DataFrames.

**Table 3. Contents of Each Cell in the Row for Net Sales**

'Net sales'	'\xa0'	'\$'	'123,897'	' '	'\xa0'	'\$'	'122,136'	' '
-------------	--------	------	-----------	-----	--------	------	-----------	-----

## 5.2 Recurrent Neural Network

Besides learning to change built in parameters in the Keras API, we had to write our own function to correctly make our LSTM cells have the memory of one. That is, we wanted the LSTM cells to put the highest importance on the one last most recent data set. In order to achieve this we had to write our own function to change the shape of the input vector. As a result, we wrote a Python method called “series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True).” This function reads in the csv file that contains specific company’s data and outputs a vector to feed into the model. For testing purposes, the script used made up data values with very obvious trends.

To validate the function, we printed out the reframed vector and checked manually against the csv file. We passed the created sample data into the python method to reframe the data to a vector that our model will take in for supervised learning. Figure 6 in Appendix C shows a few rows of the reframed data. We set the n\_in to one, which means that our LSTM cells will have memory of one. We also set the n\_out to 1 because we only wanted to test predicting one future stock price. As shown in Appendix E, the reframed vector has twelve (t-1) variables and then one (t) variable. The twelve (t-1) variables represent all the features(including a stock price) that we use to predict the one (t) variable, a future stock price.

## 6.0 TIME AND COST CONSIDERATIONS

In the first semester of working on this project, our group created a Gantt chart to illustrate our project schedule for the second semester. We managed to stay on track with our schedule for August and the first half of September, however, an unexpected change in the relationship with our industry sponsor—State Street—forced us to deviate from our planned schedule. A significant change that was made in mid-September was pivoting our project to mainly focus on

prediction instead of correlation (although we ultimately still included aspects of correlation via clustering). This had a ripple effect on our user interface creation as we had to pivot what needed to be programmed for the web application—for both the back-end and front-end. The end effect of this was that our user interface timeline was extended up until the weekend before senior design showcase, and our testing was concurrent with the addition of new features throughout the latter half of the user interface timeline. Our machine learning timeline was upheld for the most part, but this was only possible because two of our four group members worked solely on the machine learning aspect of the design (due to our decision to pivot to prediction while also including correlation). Our final delivery for the project (elevator pitch, showcase poster, and other final course deliverables) met the scheduled time constraints accurately.

As our solution was strictly software-based, there were no physical materials that needed to be purchased. A few software-related cost considerations that our team took into account were: (1) hosting our web application on cloud servers, (2) purchasing software to integrate with our web application, and (3) purchasing financial data from a market data feed. However, we opted not to pursue any of these tasks, as they were either unnecessary or added little to no marginal value, so our team ended up spending zero dollars for the implementation of our design.

## **7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN**

Our product interface is a web application that does not rely on any hardware components. As such, there were no physical safety concerns that needed to be taken into account for any stakeholders of our product. Our ethical considerations also did not need to consider any possible health/general welfare/environmental impacts for the same reason. Because of this, we focused on areas such as data security and intellectual property for our ethical design. One of our initial considerations for our product was to ensure that our product did not comprise the intellectual property (market data) of State Street—our industry sponsor. However, we ultimately scraped our own financial data from EDGAR instead of receiving data from State Street, so there was no intellectual property that needed to be protected. The data that we wished our design to protect was both the company financial data that users uploaded via CSVs and the historical stock data that users could scrape via the Alpha Vantage API. Although our application relies on several machine learning scripts that utilize the data that is both uploaded and scraped, we planned our



application design to ensure that the data itself could not be modified in any way by end users—protecting the validity and reliability of this data. Another ethical stance that we took into account given the nature of our product—predicting the future stock price of companies—is that we stressed to users that our future predictions are not meant to give professional financial advice that should be strictly followed. We do not claim to be anywhere close to perfect predictions, as the stock market is volatile and can change due to many factors—including (but not limited to) public sentiment, interest rates set by the Fed, and even presidential tweets. Our product is meant to give supplementary information that is supportive of existing financial insights.

## **8.0 RECOMMENDATIONS**

Throughout our two semester journey we encountered a number of struggles related to two distinct areas of execution; lack of proper data and the idea of correlation versus causation. We believe that any future development of this project would achieve the most constructive progress by focusing on these two areas. From the beginning, the inaccessibility of EDGAR provided a distinct obstacle to obtaining a clean set of data. Additionally, the lack of financial expertise among our team members limited our ability to pinpoint which types of data would maximize the results of our solution. This, in part, contributed to our struggles with correlation versus causation. Within a recurrent neural network, the heart of our project, dataset features are given weights to produce a formula for prediction (recall that features are the specific data, such as revenue or cost per share, that make up our financial database). This formula, while useful for creating accurate predictions, is a multi-dimensional, convoluted equation which is nearly impossible to interpret. As a result, we were unable to tell which features proved most useful to our system. The remainder of this section discusses our team's recommendations for pursuing thoughtful alterations of the solution in order to better alleviate these struggles in the future.

### **8.1 Obtaining Data**

The key to unlocking useful machine learning is access to a large, clean dataset that can provide useful insight to the problem being addressed. Our project was born from the desire to exploit EDGAR for its incredible abundance of unique data. Consequently, we spent a significant portion of our early development learning the exact nature of what data can be found in EDGAR.

It became obvious fairly early in the project that our team's skill set was not optimal for complete utilization of the database. As discussed in Section 3.1, the content within EDGAR provided two unique challenges; granularity and parsing. The forms from which we were attempting to parse data varied incredibly in both the formatting of the document and the type of data included. Many of the documents included only text, without quantitative data, describing different events in the life and growth of a company's success (or failure). Forms that included quantitative data completely lacked any form of standardization, both within a company and across companies. Thus, the keyword describing a desired data feature, such as 'revenue,' may vary from company to company and from year to year. Consequently, because of our desire to focus the bulk of our project on machine learning, not data scraping, we limited our starter dataset to the nine companies listed in sections 3.0 and 3.1. Had we understood the difficulty of procuring data from EDGAR at the start of our project, we would have dedicated less time to learning to scrape its contents. In the future of the project, we recommend developing tools that are versatile enough to comb through the chaotic depths of EDGAR.

Another aspect of data we struggled with was the value of our chosen features. After some time researching financial definitions, we chose several features that we believe are indicative of a company's success. However, there are many other measures that could be included in our model given more time or financial understanding. For example, the stock market is affected in part by the federal interest rate. This is a quantitative datapoint that could be included as a feature in the model. Public sentiment is also a key factor in understanding the stock market. If a company faces some public scandal, or develops a revolutionary new tool that the public adopts, their stock price may be affected drastically without any effect on the features chosen for our model. In order to quantify public sentiment, you would likely need some form of artificial intelligence to monitor social media for keywords related to specific companies. This AI would then have to translate those keywords into a usable form for the RNN. It is also possible much of this information would be available in EDGAR. Many documents within EDGAR, like quarterly reports, are periodic and filed at regular intervals. Other documents are filed when specific events happen in the life of a company; for example, acquiring another company. These events are typically described through documents that are mostly text and require translation to a usable form. While gauging public sentiment would be an extremely valuable feature to include in the

model, it was outside the scope of both our project and team skill set. We highly recommend finding a useful method for incorporating more varied data into the prediction model.

## **8.2 Understanding Correlations**

When we were first examining the motivation for exploring EDGAR, we initially concentrated on the concept of correlation. The data within EDGAR is unique because it has both a wide variety of data for individual companies and data for numerous companies in a single accessible location. Immediately, intuition led us to the idea of using this data to see how companies relate to each other in the financial sphere. While we still incorporated this idea into our final product, it did not end up the focal point of our solution; prediction became our target. Based on the data issues we discussed in section 8.1, we chose prediction to better fit our goal of proving the potential applications of exploiting EDGAR. This, in turn, led to the development of the RNN. RNNs are incredibly useful for producing accurate data analysis, but not for producing interpretable data analysis. With the model we developed, we essentially fed our data into a black box and read the output. There were several parameters we could tweak, such as number of epochs, but no result that specifically indicated which features were most effective for the prediction. Using our clustering algorithm, we were able to correlate certain companies, but we had only intuition to back up our understanding of the groups. Together, our RNN and clustering suggest that data from EDGAR can be useful for both correlation and prediction. However, we have two recommendations for ways to show that these correlations hold significant meaning beyond the obvious result. First, we recommend that feature correlation be added to the model. Second, we believe that having financial expertise, in either the form of an outside collaborator or a dedicated team member, would improve the usefulness of each feature included in the model.

Feature correlation relies on the same principle as our clustering algorithm. Examining trends of different features over time provides insight into what features grow and decay in similar ways. One useful way of discovering meaning behind which features may hold particular value for prediction is to correlate those features with either the stock price, our target label, or other features. If certain features have either a positive or negative correlation with other features, they may reveal the reason the RNN behaves as it does. This could be accomplished using a modified

version of our existing K-Means algorithm. In fact, it is likely that we would have incorporated this concept into the model ourselves if we had pursued this solution for a longer period of time.

Perhaps our biggest struggle towards the start of development for this project was our lack of financial expertise as a team. Understanding the machinations of how the stock market operates is a formidable task that requires years of dedication. Each member of our team had studied only engineering prior to this course. Together, we began research into the financial world based off our initial inquiries of the EDGAR database. While we feel that we made significant progress in developing the understanding required to select specific features for the model, it is probable that our lack of financial mastery hindered the fullest potential of our solution. If we were to repeat this project, we likely would have searched for a team member that had taken courses related to the world of finance. That team member would have been able to guide our intuition and ground the model in a realistic understanding of financial markets. This would have significantly increased the team's ability to explore EDGAR, choose relevant financial features, and begin development of the model at an earlier stage with more lucrative data. It is also possible that having an outside consultant to refer to in situations of financial discovery could have accomplished the same results. Moving forward, we believe that having access to a team member with financial education would provide the foundation for obtaining the data described in section 8.1 and developing a more comprehensive model.

## **9.0 CONCLUSIONS**

In this report, we described our engineering design problem of creating a tool that utilizes public financial data to aid users in making investment decisions. We discussed the different modules that make up our final product, as well as the design decisions we made to achieve a reasonable level of accuracy with insufficient data. We covered the changes our project underwent during the design process, as we tuned the model while also taking into account the limited time and resources we had available. Finally, we looked back on our work during the past year, and outlined how another group could improve upon our initial project by making use of more data and discovering ways to better organize and interpret their findings.

Overall, we were not able to determine to what extent the contents of a company's quarterly reports affect the stock price of that company. However, we were still able to make reasonable predictions with an RNN by utilizing clustering to improve accuracy despite our severe lack of well-formed data. We were able to create a scraper from scratch that was able to obtain feature values from hundreds of documents despite the documents having no universal standards for formatting. In addition, we made use of historical stock data in a unique way, by clustering time series of the input companies and combining these findings with EDGAR data and retraining the RNN with an input layer that varies based on the user input. Finally, we were able to integrate each of these scripts to create a user-friendly web application that clearly displays the results of each module. There are many additional factors that could've been taken into account in the model, but our project was a first step in an area with great potential for further exploration.

For future projects dealing with similar topics, the importance of data cannot be understated. Specifically, the quantity of data must be increased, and the group members must have a better understanding of the financial terms used in order to have greater knowledge of how best to form the machine learning model. Better utilization of all of the documents present on EDGAR would greatly bolster the dataset, and incorporation of data from Google Trends, Twitter, and the Federal Reserve will take into larger variables at play that can have an effect on either a single company's stock or the stock market as a whole. Having access to someone with financial knowledge would also allow group members to better interpret data, rather than simply feeding it all into an RNN and hoping the prediction is accurate. Each of the modules in our system serve as valuable starting points. Greater use of publicly available data can take our goal of discovering correlations and making predictions from financial data to the next level.

## REFERENCES

- [1] *Filings & Forms* [Online]. Available: <https://www.sec.gov/forms>
- [2] Python Data Analysis Library — pandas, [pandas.pydata.org](https://pandas.pydata.org), pandas 0.24.2, 2019
- [3] Requests: HTTP for Humans, [requests.kennethreitz.org](https://requests.kennethreitz.org), Requests 2.22.0, 2019
- [4] Beautiful Soup Documentation, [Crummy.com](https://crummy.com), Beautiful Soup 4.4.0, 2016.
- [5] API Documentation, [Alphavantage.co](https://alpha.vantage.co), Alpha Vantage 2.1.1, 2019.
- [6] Akhilesh G. (2019, April 30) *Adjusted Closing Price* [Online]. Available: [https://www.investopedia.com/terms/a/adjusted\\_closing\\_price.asp](https://www.investopedia.com/terms/a/adjusted_closing_price.asp)

## **APPENDIX A – SPECIFICATIONS TABLE**

## APPENDIX A – SPECIFICATIONS TABLE

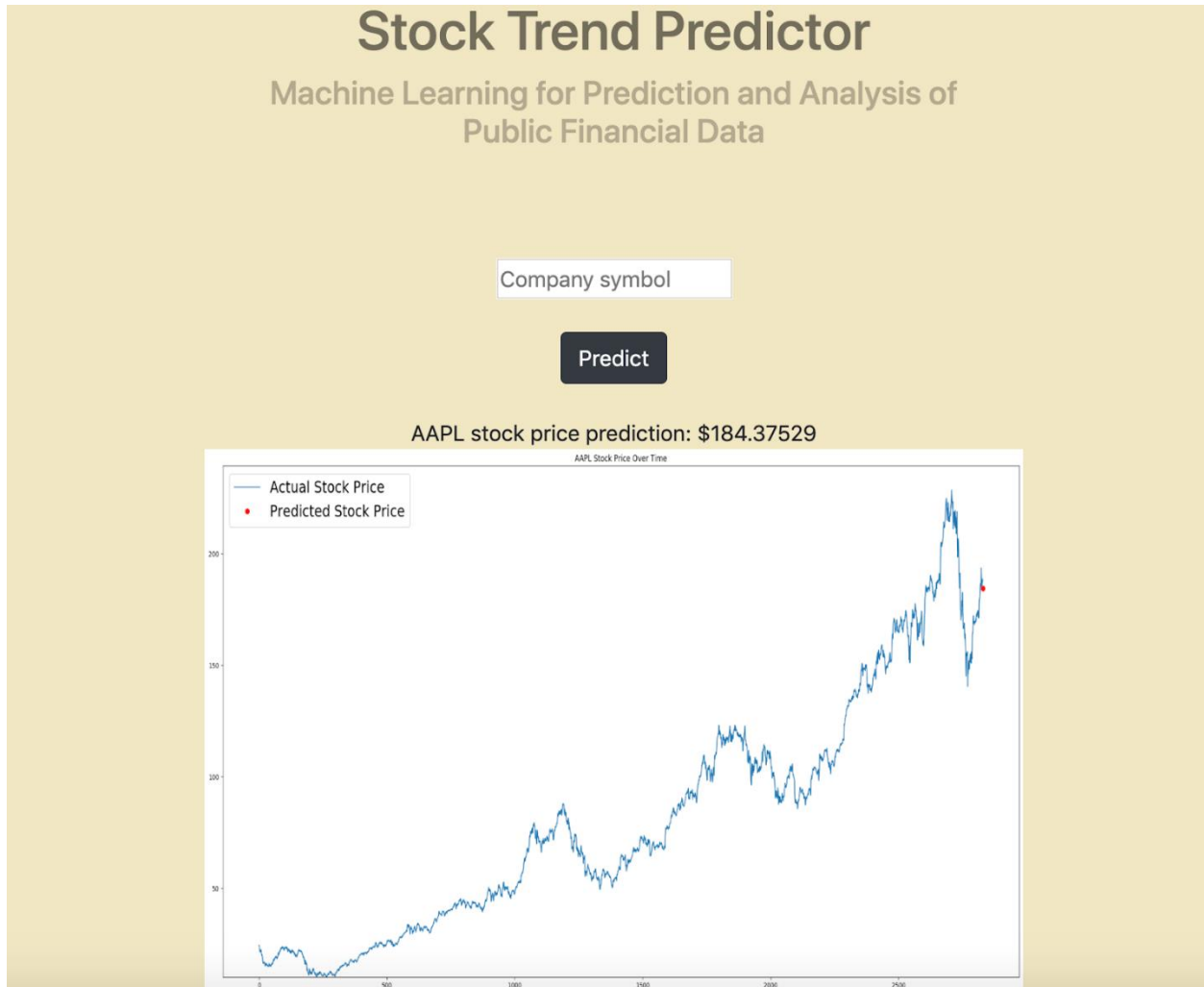
**Table 4. Specifications Table**

Requirement	Description	Specification
Data acquired, companies	We have acquired data for a small set of companies to be stored on our back end	9 companies
Data acquired, time period	Our data for each company will span a period of 10 years	2008 to 2018
Number of features for each company	Each dataset has a set of numerical features, as well as some descriptors such as company name, year, and quarter	11 Features
Scraper Language		Python
Back End Language		Javascript, NodeJS/Express
RNN Language		Tensorflow, Keras
Front End Language		Javascript, HTML, CSS
Dataset Format		CSV
Web App Deployment		Local
Wireframes	Before coding our UI, we want to use wireframes to inform the design	One or more wireframes for each page of the web app
UI Navigation	The top of each page will include a toolbar.	Mouse navigation
UI Look and Feel	We want our UI to be simple to use. Additionally we want our design to look clean and not too crowded	N/A
Landing Page	This page should present a simple first impression of our web app	
About Us Page	This page should include a description of the project as well as our project team	<i>Features:</i> Pictures of each team member, text descriptions
Solution Display Page	This page should present the results of our prediction	<i>Features:</i> Stock price graph, prediction result
EDGAR Page	This page should give users an introduction to the EDGAR database	<i>Features:</i> Text description, hyperlink to EDGAR
Data Upload Tutorial	This page should guide users through the upload process	<i>Features:</i> Text description, example image
Data Storage	When users upload data, we must store it on our back end	Stored in project directory on server



## **APPENDIX B – USER INTERFACE SCREENSHOTS**

## APPENDIX B – USER INTERFACE SCREENSHOTS



**Figure 4. Prediction Page of Web Application**

# Clustering

## Cluster Tutorial

Steps to cluster data:

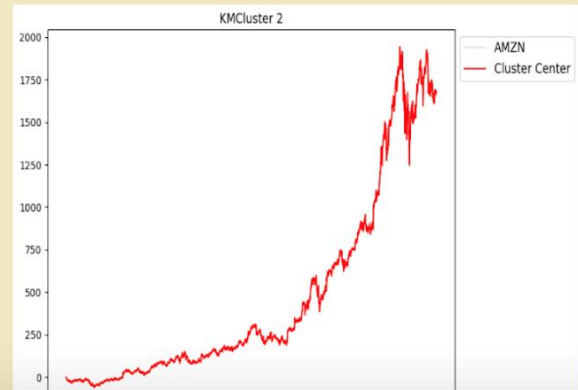
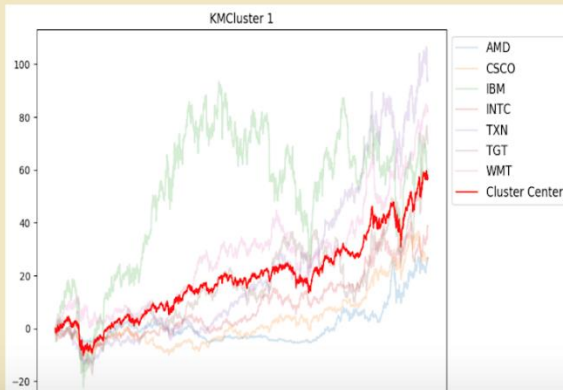
1. Enter a company's ticker symbol
2. Click 'scrape' to enter the stock info into our CSV
3. Click cluster to see the cluster analysis of related companies

## Cluster Here

Scrape

Cluster

Reset



**Figure 5. Clustering Page of Web Application**

## **APPENDIX C – RANDOMLY GENERATED DATA**

## APPENDIX C – RANDOMLY GENERATED DATA

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	\
1	8683.0	11918.0	2899.0	8186.0	4083.0	12989.0	
2	8302.0	12056.0	2912.0	8178.0	3936.0	11192.0	
3	8693.0	12201.0	2893.0	8254.0	3980.0	12539.0	
4	8380.0	13741.0	2847.0	7863.0	3978.0	11249.0	
5	8686.0	12428.0	2642.0	8184.0	3691.0	12204.0	
	var7(t-1)	var8(t-1)	var9(t-1)	var10(t-1)	var11(t-1)	var12(t-1)	\
1	11708.0	3015.0	18418.0	6048.0	8990.0	8095.0	
2	11110.0	3216.0	19862.0	6258.0	9198.0	7786.0	
3	11725.0	2990.0	17589.0	6447.0	8650.0	7331.0	
4	11996.0	3172.0	18262.0	6533.0	9514.0	7983.0	
5	12049.0	2981.0	19613.0	6382.0	9484.0	8966.0	
	var12(t)						
1	7786.0						
2	7331.0						
3	7983.0						
4	8966.0						
5	8484.0						

Figure 6. Reframed Randomly Generated Data